CSIT 6910 Independent Project

A survey of big data architectures for handling massive data

Jordy Domingos - jordydomingos@gmail.com
Supervisor : Dr David Rossiter

Content Table

1. Introduction

a. Context and motivation

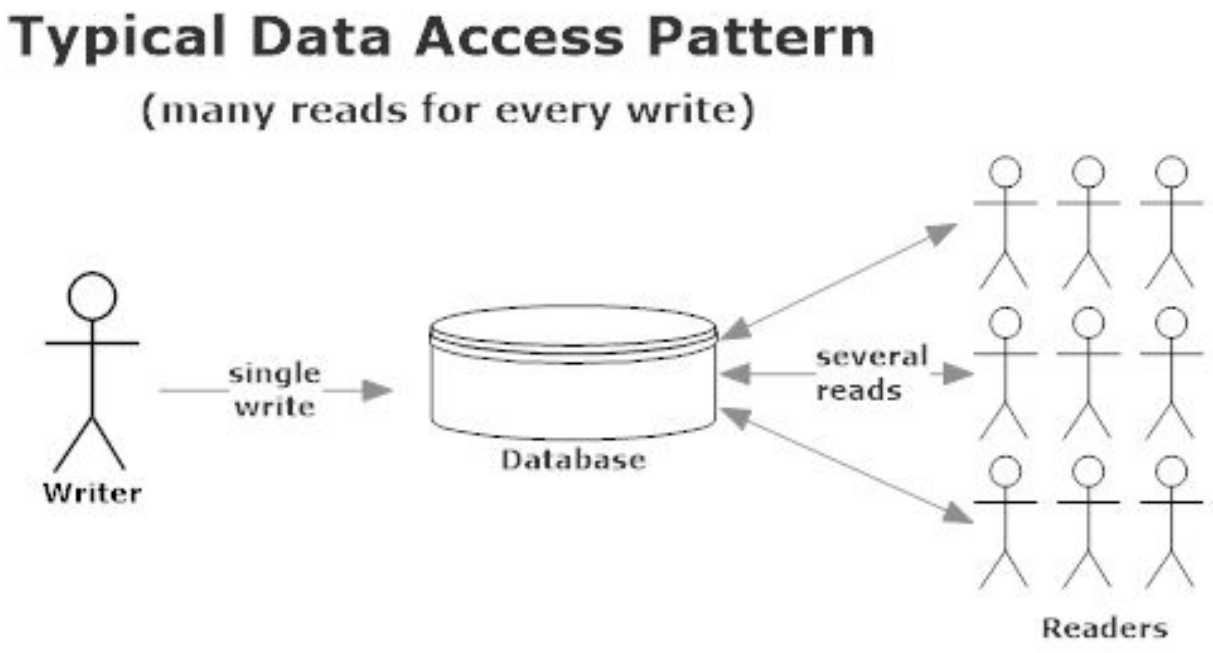
As a Master's student that is not going to continue his studies toward a Phd degree, I needed to find an area in which I would like to work in the close future. Recently, I took a Data Mining course given by Dr Lei Chen and I found that area really interesting. After further thinking, I decided that I would put most of my energy into learning more concepts and techniques related to the Big Data field.
I managed to secure an internship at Airbus as a Big Data Engineer starting in september 2015 and while waiting patiently for the time to come, I decided to work on an small project related to the Big Data field and the work that I will have to do during my internship.

b. Project description

The goal of the project is to learn more about the big data ecosystem by a survey of big data architecture that could possibly handle a massive amount of data.
To build this architecture, I am going to use loop sensor data that were collected for the Glendale on ramp for the 101 North freeway in Los Angeles.

The sensor is close enough to the stadium to see unusual traffic after a Dodgers game, but not so close and heavily used by game traffic so that the signal for the extra traffic is overly obvious.
These loop sensor measurements were obtained from the Freeway Performance Measurement System (PeMS), "http://pems.eecs.berkeley.edu"
The main goal of the project is to build an architecture that could handle a massive amount of data.

To achieve my goal I start by comparing the different tools that are used in the industry.
Then I will select the tools that fit the best my needs. The dataset selected itself is not big enough to be considered as "massive" but we could just imagine that we had **1 TeraByte** of similar data. The structure and attributes of the data set will be used as a base to optimize the architecture if needed



1. *Typical data access pattern.*

Regarding the data access pattern, I assumed that the data collected would be inserted to the database and then never touched again. The readers represent the people who are trying to extract knowledge from the stored data.

c. Technical environment

In this project, everything is conceptual and will be explained. As everything in this project is new for me, this report will contain all the thoughts that led me toward the final solution. Each step is an incremental move toward it.

2. Data set description

This loop sensor data was collected for the Glendale on ramp for the 101 North freeway in Los Angeles. It is close enough to the stadium to see unusual traffic after a Dodgers game, but not so close and heavily used by game traffic so that the signal for the extra traffic is overly obvious.

NOTE: This is an on ramp near the stadium so event traffic BEGINS at or near the END of the event time.

The observations were taken over 25 weeks, 288 time slices per day (5 minute count aggregates).

The goal is to predict the presence of a baseball game at Dodgers stadium

Attribute Information:
1. Date: MM/DD/YY
2. Time: (H)H:MM (military time)
3. Count: Number of cars measured for the previous five minutes
Rows: Each five minute time slice is represented by one row

For .events file:
1. Date: MM/DD/YY
2. Begin event time: HH:MM:SS (military)
3. End event time: HH:MM:SS (military)
4. Game attendance
5. Away team
6. W/L score

You can dowload this dataset at : https://archive.ics.uci.edu/ml/datasets/Dodgers+Loop+Sensor

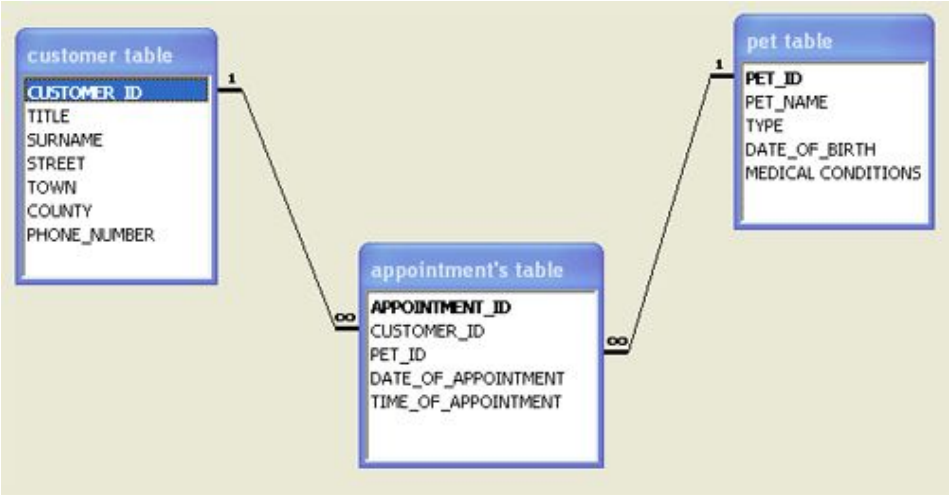
3. Big Data Technologies

I organized this overview as an incremental journey from the worst choices to the best ones.
In this journey I haven't consider the use of flat files to store the data because it is obvious that storing 1 TeraByte using that technique would lead to a big number of issues in term of scalability. Therefore we will only look at better solutions.

a. Relational database management system (RDBMS)

i. Relational Database

A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables.[1]



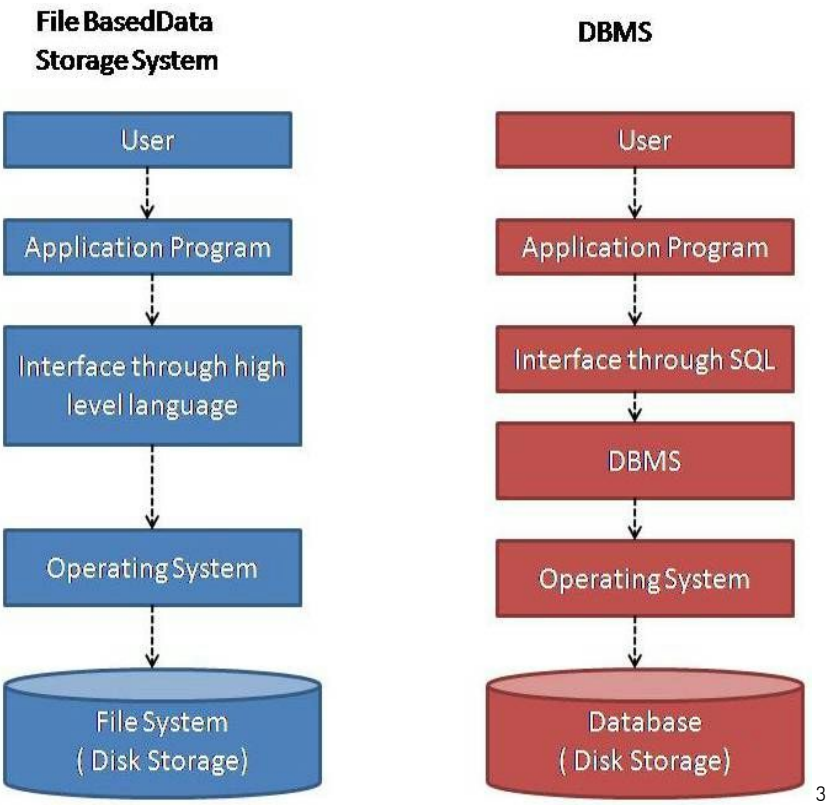
2. *Example of relation between 3 entities.*

ii. Relational Database Management System

RDBM systems are the first components that I studied to build my architecture.
A relational database management system (RDBMS) is a program that lets you create, update, and administer a relational database.

[1] http://searchsqlserver.techtarget.com/definition/relational-database

RDBMS's use the Structured Query Language (SQL) to access the database, although SQL was invented after the development of the relational model and is not necessary for its use.[2]



[3]

3. *Comparison between file based storage and DBMS.*

iii. ACID Compliance

ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee that database transactions are processed reliably.[4]

In the context of databases, a single logical operation on the data is called a transaction. For example, a transfer of funds from one bank account to another, even involving multiple changes such as debiting one account and crediting another, is a single transaction.

[2] http://searchsqlserver.techtarget.com/definition/relational-database-management-system
[3] http://www.careerbless.com/db/rdbms/c1/intdbms.php
[4] https://en.wikipedia.org/wiki/ACID

- Atomicity

In an atomic transaction, a series of database operations either all occur, or nothing occurs.

- Consistency

Requirement that any given database transaction must change affected data only in allowed ways.

- Isolation

Determines how transaction integrity is visible to other users and systems.

- Durability

Guarantees that transactions that have committed will survive permanently.

iv. Pros and cons

RDBMS have many advantages over a flat file:

- Avoids data duplication
- Avoids inconsistent records
- Easier to change data
- Easier to change data format
- Data can be added and removed easily
- Easier to maintain security.

But they also have major issues.

- RDBMSs alone don't make data access faster but only simpler.
- Not all data requires ACID compliant data stores.
- To implement ACID, there are tradeoffs to make that limit scalability of traditional systems

Let's see how what we can use to improve these points.

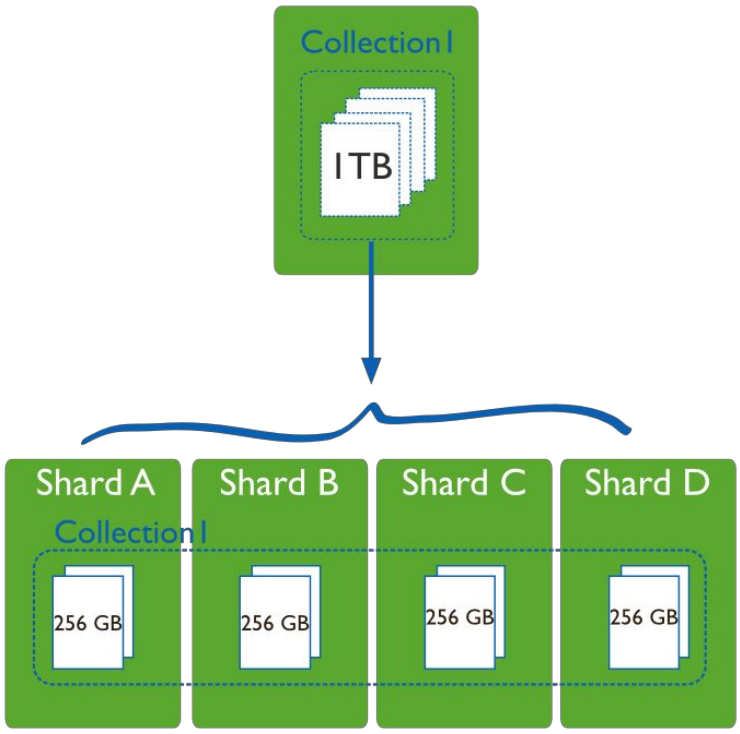
b. Sharding

Database systems with large data sets and high throughput applications can challenge the capacity of a single server. High query rates can exhaust the CPU capacity of the server. Larger data sets exceed the storage capacity of a single machine. Finally, working set sizes larger than the system's RAM stress the I/O capacity of disk drives.[5]

Sharding addresses the challenge of scaling to support high throughput and large data sets:

- Sharding reduces the number of operations each shard handles. Each shard processes fewer operations as the cluster grows. As a result, a cluster can increase capacity and throughput horizontally. For example, to insert data, the application only needs to access the shard responsible for that record.

- Sharding reduces the amount of data that each server needs to store. Each shard stores less data as the cluster grows. For example, if a database has a 1 terabyte data set, and there are 4 shards, then each shard might hold only 256GB of data.

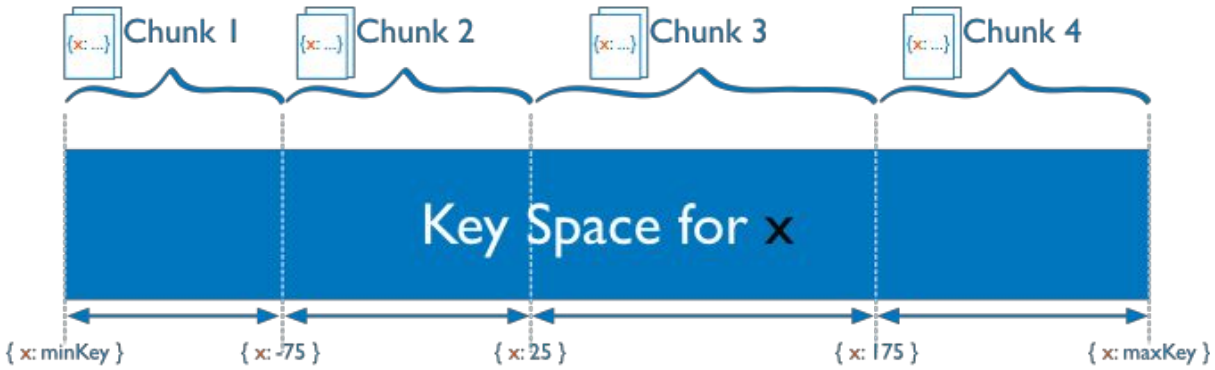


4. *Shard creation example.*

[5] http://docs.mongodb.org/manual/core/sharding-introduction

i. Range Based Sharding

To understand range based sharding, consider that the documents are divided by range chunks.
We attribute a key to each document in memory from minKey to maxKey and the value associated is a value that leads us to the right chunk to access the data faster.



5. *Range Based Sharding example.*

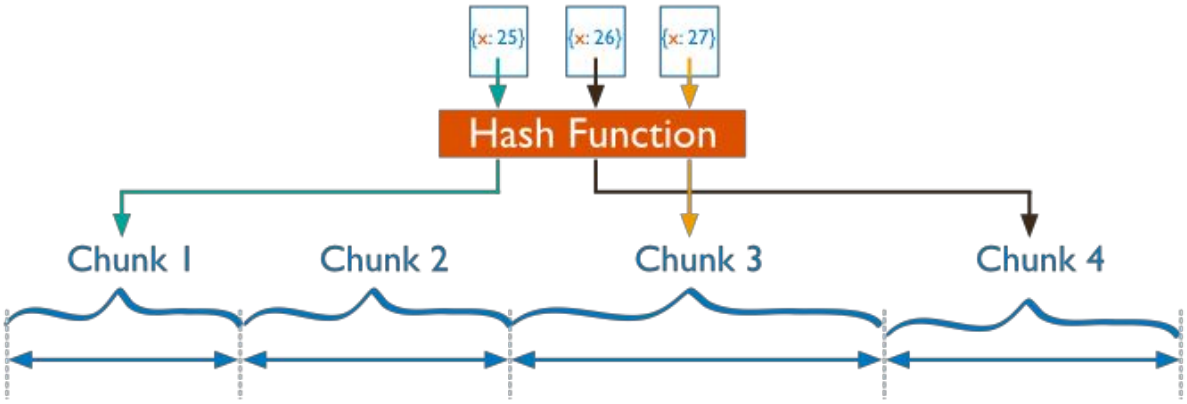
ii. Hash Based Sharding

A hash function is any function that can be used to map digital data of arbitrary size to digital data of fixed size. The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes. One use is a data structure called a hash table, widely used in computer software for rapid data lookup.[6]

6. *Hashing example.*

[6] https://en.wikipedia.org/wiki/Hash_function

In hash based sharding, each document has a key that is used by an hashing function to retrieve the correspondent chunk.

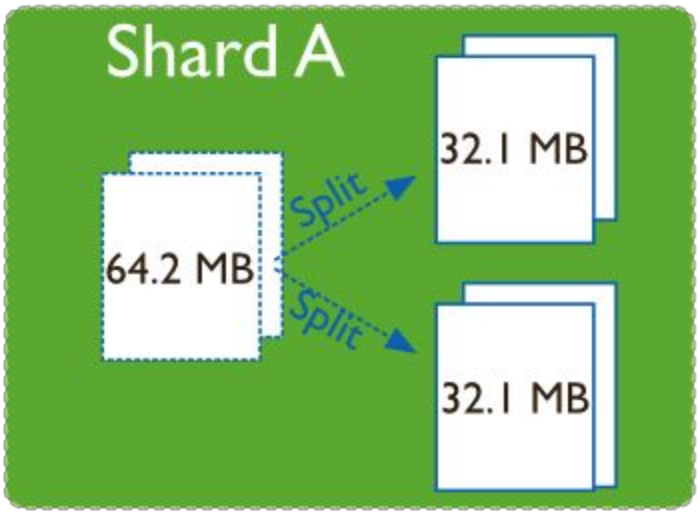


7. *Hash Based Sharding example.*

RDBMSs that support sharding have to support these two basic operations.

iii. Splitting

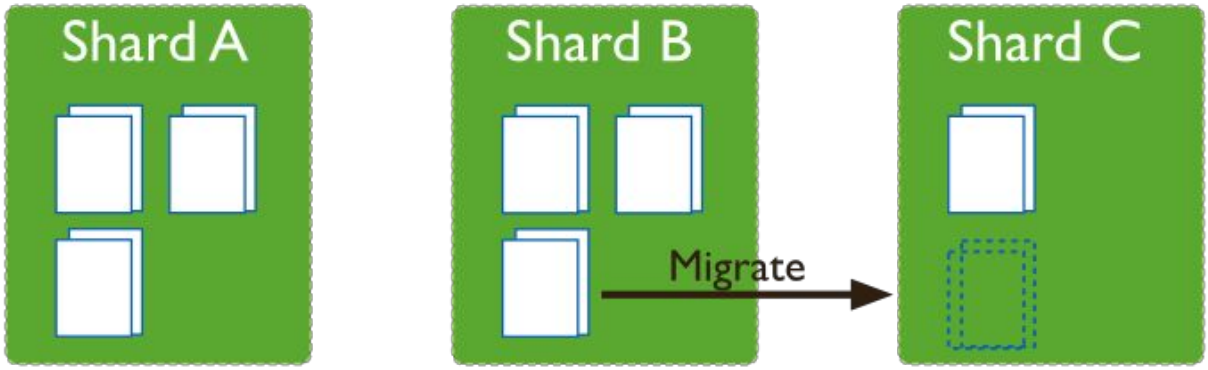
When a shard comes to big, it should split to two smaller shards.



8. *Shard splitting example.*

iv. Balancing

When some shards are unevenly full, the balancing send some documents to other shards.



9. *Document migration example.*

v. Pros and cons

Sharding is a good solution to retrieve documents coupled to RDBMS but unfortunately it does not solve all problems.
Indeed we still have to deal with ACID compliance tradeoffs.
Let's see how we can solve these problem.

c. NoSQL

A NoSQL (originally referring to "non SQL") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.[7]

Motivations for this approach include simplicity of design; simpler "horizontal" scaling to clusters of machine, which is a problem for relational databases; and finer control over availability. The data structures used by NoSQL databases (e.g. key-value, graph, or document) differ slightly from those used by default in relational databases, making some operations faster in NoSQL and others faster in relational databases. The particular suitability of a given NoSQL database depends on the problem it must solve.

In our case, there is no relational relationship between the events. Therefore the relationships are not needed. This approach solves RDBMS scalability issues because it abandons ACID compliance in favor of scalability.

i. CAP theorem

The CAP theorem, also known as Brewer's theorem, states that it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees.

- **Consistency** (all nodes see the same data at the same time)
- **Availability** (a guarantee that every request receives a response about whether it succeeded or failed)
- **Partition tolerance** (the system continues to operate despite arbitrary partitioning due to network failures)[8]

For our project we should focus on availability and partition tolerance. We don't need to focus on consistency because the data won't be changed and don't evolved in time

[7] https://en.wikipedia.org/wiki/NoSQL
[8] https://en.wikipedia.org/wiki/CAP_theorem

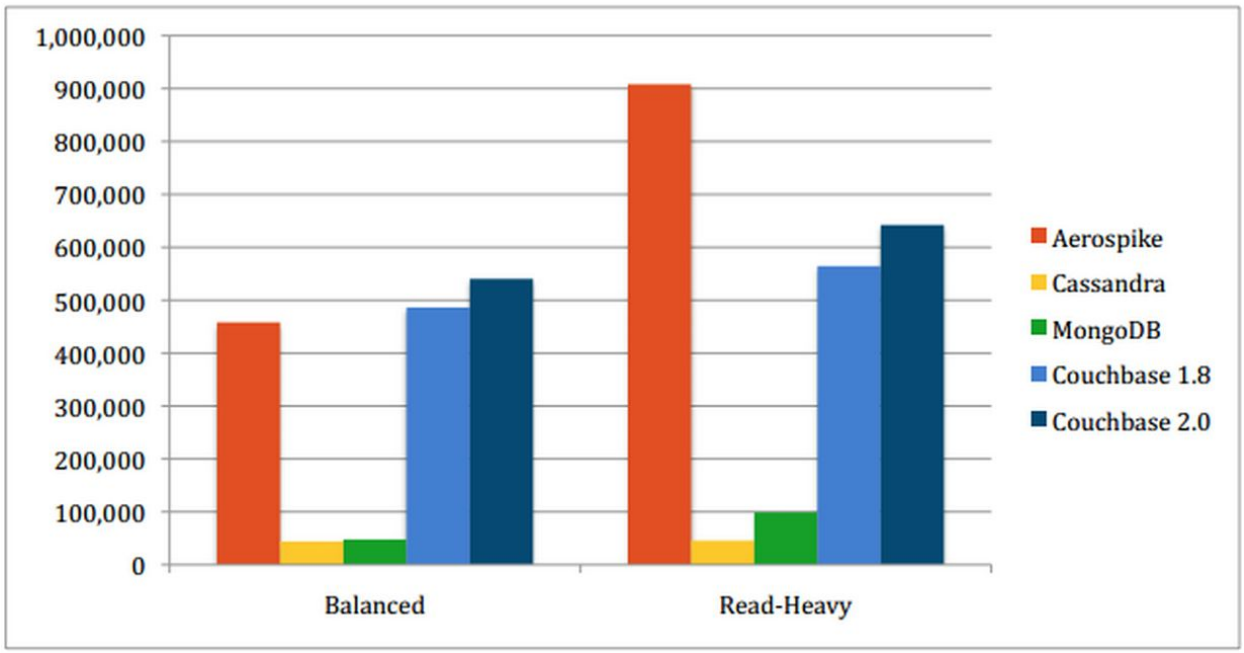
10. *CAP theorem and DataBase attached to their different properties.*

ii. Speed Requirements

Knowing that our needs are really simple (Reads required to perform our future computation over the data) I decided to back this choice using the result on http://www.thumbtack.net/ that released two benchmark whitepapers with results from the comparison of several key-value stores: Ultra-High Performance NoSQL Benchmarking: Analyzing Durability and Performance Tradeoffs and NoSQL Failover Characteristics: Aerospike, Cassandra, Couchbase, MongoDB[9].

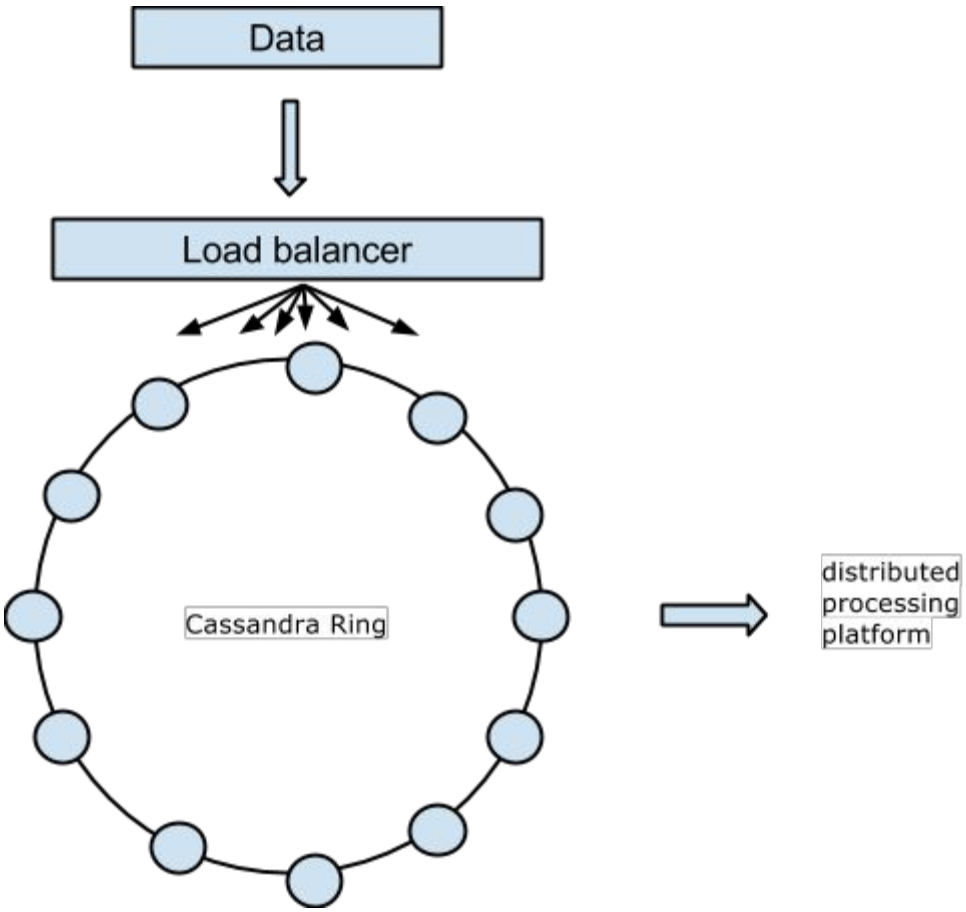
The NoSQL databases tested were Aerospike[10], Cassandra[11], Couchbase[12] (1.8 and 2.0), and MongoDB[13]. The first is a commercial product, and the last is a document data store not a key-value store.
All databases were optimized using recommendations from the vendors backing them. The test systems used SSD storage rather than rotating disks. The white papers contain detailed information regarding the methodology used, the client and workload configuration, the hardware configuration, etc.
You can find the most useful test below that confirm that **Cassandra** might be the best choice.

[9] http://www.thumbtack.net/whitepapers/
[10] http://www.aerospike.com/
[11] http://cassandra.apache.org/
[12] http://www.couchbase.com/
[13] https://www.mongodb.org/



11. *Maximum Throughput - In Memory Data Set*

4. Final architecture



12. *My Big Data architecture.*

i. Load balancer

A Load balancer distributes workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units or disk drives. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. In our architecture, the load balancer will divide the data to the multiple nodes present in the Cassandra Ring.

ii. Cassandra Ring

As discussed previously, we decided to pick Cassandra as NoSQL database to store our data.

Cassandra is designed to handle big data workloads across multiple nodes with no single point of failure. Its architecture is based on the understanding that system and hardware failures can and do occur. Cassandra addresses the problem of failures by employing a peer-to-peer distributed system across homogeneous nodes where data is distributed among all nodes in the cluster.

Cassandra stores replicas on multiple nodes to ensure reliability and fault tolerance.
All replicas are equally important; there is no primary or master replica.

In this architecture, each node on the cassandra ring represents a server.
After some research, I decided to use Dell PowerEdge R610 servers due to their specification as they are largely able to accomplish the wanted tasks.

Feature	Technical Specification	
Form Factor	1U rack	
Processors	Quad-core or six-core Intel® Xeon® processors 5500 and 5600 series	
Processor Sockets	2	
Front Side Bus or HyperTransport	Intel® QuickPath Interconnect (QPI)	
Cache	4MB and 8MB	
Chipset	Intel® 5520	
Memory[1]	Up to 192GB (12 DIMM slots): 1GB/2GB/4GB/8GB/16GB DDR3 up to 1333MHz	
I/O Slots	**2 PCIe G2 slots + 1 storage slot:** Two x8 slots One storage x4 slot	
RAID Controller	**Internal:** PERC H200 (6Gb/s) PERC H700 (6Gb/s) with 512MB battery-backed cache; 512MB, 1GB Non-Volatile battery-backed cache SAS 6/iR PERC 6/i with 256MB battery-backed cache	**External:** PERC H800 (6Gb/s) with 512MB of battery-backed cache; 512MB, 1GB Non-Volatile battery-backed cache PERC 6/E with 256MB or 512MB of battery-backed cache **External HBAs (non-RAID):** 6Gbps SAS HBA SAS 5/E HBA LSI2032 PCIe SCSI HBA
Drive Bays	Internal hard drive bay and hot-plug backplane. Up to six 2.5" SAS, or SSD Drives	
Maximum Internal Storage	Up to 12TB	
Hard Drives[1]	**Hot-plug Hard Drive Options:** 2.5" SAS SSD, SATA SSD, SAS (10K, 15K), nearline SAS (7.2K), SATA (7.2K)	
Communications	Two dual port embedded Broadcom® NetXtreme® II 5709c Gigabit Ethernet NIC with failover and load balancing. Optional 1GBe and 10GBe add-in NICs Broadcom® NetXtreme® II 57711 Dual Port Direct Attach 10Gb Ethernet PCI-Express Network Interface Card with TOE and iSCSI Offload Intel® Gigabit ET Dual Port Server Adapter and Intel® Gigabit ET Quad Port Server Adapter Dual Port 10GB Enhanced Intel Ethernet Server Adapter X520-DA2 (FCoE Ready for Future Enablement) Brocade® CNA Dual-port adapter Emulex® CNA iSCSI HBA stand up adapter OCE10102-IX-D Emulex® OCE10102-IX-DCNA iSCSI HBA stand-up adapter **Optional Add-In HBA's:** Brocade® 8 GB HBAs	
Power Supply	Two hot-plug high-efficient 502W Energy Smart PSU or two hot-plug 717W High Output PSUs **Uninterruptible Power Supplies:** 1000W-5600W 2700W-5600W High Efficiency Online Extended Battery Module (EBM) Network Management Card	

13. *Server specification.*

iii. Distributed processing platform

The data stored in the servers could be handle by a distributed processing platform to perform computation on it faster.
This is not mandatory for our needs but I included it to open future discussions about the following architecture.

5. Further thinking

As we said, the architecture that we built is good to store and retrieve the data. But what if we also want to do some computations on it ?

a. Hadoop

Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are commonplace and thus should be automatically handled in software by the framework.
The core of Apache Hadoop consists of a storage part (Hadoop Distributed File System (HDFS)) and a processing part (MapReduce). Hadoop splits files into large blocks and distributes them amongst the nodes in the cluster. To process the data, Hadoop MapReduce transfers packaged code for nodes to process in parallel, based on the data each node needs to process. This approach takes advantage of data locality—nodes manipulating the data that they have on hand—to allow the data to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are connected via high-speed networking.[14]

b. Distributed file system

A Distributed file system (DFS) is a method of storing and accessing files based in a client/server architecture. In a distributed file system, one or more central servers store files that can be accessed, with proper authorization rights, by any number of remote clients in the network. Much like an operating system organizes files in a hierarchical file management system, the distributed system uses a uniform naming convention and a mapping scheme to keep track of where files are located. When the client device retrieves a file from the server, the file appears as a normal file on the client machine, and the user is able to work with the file in the same ways as if it were stored locally on the workstation.

When the user finishes working with the file, it is returned over the network to the server, which stores the now-altered file for retrieval at a later time.

[14] https://en.wikipedia.org/wiki/Apache_Hadoop

Distributed file systems can be advantageous because they make it easier to distribute documents to multiple clients and they provide a centralized storage system so that client machines are not using their resources to store files.[15]

Open a file
Check status of a file
Close a file
Read data from a file
Write data to a file
Lock a file or part of a file
List files in a directory
Create/delete a directory
Delete a file
Rename a file
Add a symlink to a file

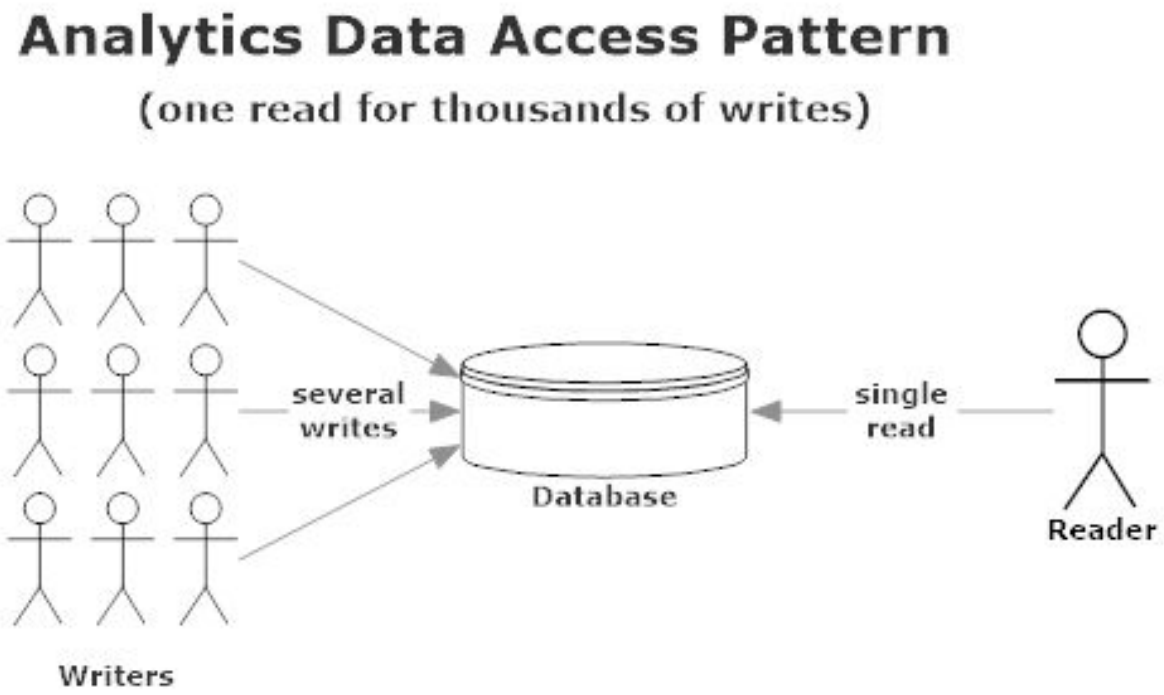
14. File system interface.

The principal functionalities of a Distributed file System are :

- Data sharing among multiple users
- User mobility
- Location transparency
- Backups and centralized management

Therefore, a good DFS is a very important piece in a Big Data architecture with less readers and writers.

[15] http://www.webopedia.com/TERM/D/distributed_file_system.html



15. Analytics data access pattern

.

c. Map/Reduce

MapReduce is a software framework that allows developers to write programs that process massive amounts of unstructured data in parallel across a distributed cluster of processors or stand-alone computers.

I won't go into the implementation details but readers must be aware that this technique can tremendously reduce the amount of computation required to perform big distributed task.

6. Conclusion

This project was really useful in order to discover the Big Data field in an architect's point of view. I am now way less stressed regarding my future internship because I know that I acquired a solid background in the field.

I would like to thank Dr Rossiter for his huge trust for that project.

7. Appendix

a. Minutes of the 1st meeting

Date :

Monday 8 June

Time :

2:00 pm

Place :

Room 3512

Attending :

Jordy Domingos
Dr. David Rossiter

Recorder :

Jordy Domingos

Approval of minutes

Previous minute approved.

Discussion Items

Detailed description of the project.
Choice of the first objectifs.
Targets :

- Understand RDBMS pros and cons + Sharding

Meeting adjournment and next meeting

The meeting was adjourned at 2:30 PM. The next meeting will be held in 3 weeks.

b. Minutes of the 2nd meeting

Date :

Monday 22 June

Time :

2:00 pm

Place :

Room 3512

Attending :

Jordy Domingos
Dr. David Rossiter

Recorder :

Jordy Domingos

Approval of minutes :

Previous minute approved.

Discussion Items :

Description of the work done (RDBMS & Sharding)
Targets for the next meeting :

- NoSQL & DFS

Meeting adjournment and next meeting :

The meeting was adjourned at 2:00 PM. The next meeting will be held in July.

c. Minutes of the 3rd meeting

Date :

Monday 6 Juillet

Time :

12 midday

Place :

Room 3512

Attending :

Jordy Domingos
Dr. David Rossiter

Recorder :

Jordy Domingos

Approval of minutes ;

Previous minute approved.

Discussion Items ;

Description of the work done on NoSQL & DFS
Targets for the next meeting :

- Finish No SQL and start Map Reduce + final design

Meeting adjournment and next meeting ;

The meeting was adjourned at 12:30 PM. The next meeting will be held in 2 weeks.

d. Minutes of the 4th meeting

Date :

Tuesday 21 Juillet

Time :

3:30 pm

Place :

Room 3512

Attending :

Jordy Domingos
Professor David Rossiter

Recorder :

Jordy Domingos

Approval of minutes :

Previous minute approved.

Discussion Items :

Description of the final architecture.

Meeting adjournment and next meeting :

The meeting was adjourned at 4:00 PM. This was the last meeting..